Sparse Non-linear System Identification in Robotics Using Meta-Learning Approaches

Thesis

Submitted in complete fulfilment of the requirements of

BITS F421T Thesis

BY

ARUNABH SINGH

2020AATS0403H

Under the supervision of

Guanya Shi

(Assistant Professor, RI, CMU)

Dr. Joyjit Mukherjee

(Assistant Professor, EEE Dept, BITS Pilani)

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF BITS F421T: THESIS

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(DECEMBER, 2023)

ACKNOWLEDGMENTS

I want to thank my supervisors, Guanya Shi and Dr Joyjit Mukherjee, for their constant support, guidance, and ease of accessibility throughout the thesis. They have been supportive and open to discussing new ideas and project details, ranging from theoretical discussions to implementation details. I also thank them for the time they invested in numerous discussions on addressing the problem and helping me develop the required academic and practical skill sets. Further, I would like to take this opportunity to thank BITS Pilani, Hyderabad Campus, for allowing me to pursue a thesis on my topic of interest and providing me with the essential opportunity to develop my research aptitude.

Birla Institute of Technology & Science, Pilani
Hyderabad Campus

03/12/23

Certificate

This is to certify that the thesis report entitled "Sparse **Non-linear System Identification in Robotics Using Meta-Learning Approaches"**, submitted by Mr Arunabh Singh (ID No. 2020AATS0403H) in complete fulfilment of the requirements of the course BITS F421T, embodies the work done by him under my supervision and guidance.

Guanya Shi

(Guanya Shi, Assistant Professor)
Date: 03/12/2023

(Joyjit Mukherjee, Assistant Professor)
Date: 03/12/2023

ABSTRACT

Learning-based quadrotor controller models like the Neural Fly have shown precise trajectory tracking under different real-time wind conditions. Neural Fly employs domain adversarially invariant meta-learning to learn the underlying shared representation of wind dynamics in different wind conditions and then employs it with an adaptive controller to fine-tune the dynamics model for real-time wind patterns, utilizing it to control the quadrotor for precise tracking. However, it uses a partial physics-based model of the quadrotor dynamics with only the residual term being learnt. Physics-based system models for control are often not a good representation of complex nonlinear systems as they overlook the cross-interaction terms in the dynamics and are computationally heavy.

We believe that data-driven system identification models are better suited for the control of such complex systems with nonlinear interactions. In the thesis presented, we have attempted to develop novel sparse nonlinear system identification methods that are generalisable and adaptable to changing system dynamics under varying environmental influences. We deploy our algorithm to model the partial dynamics for a quadrotor under different wind conditions. We have used the Neural Fly data to train and test our models. We have also shown a comparison with the state-of-the-art SINDy (Sparse Identification of Nonlinear Dynamics) algorithm which is widely used for sparse system identification.

CONTENTS

Chapter – 1
Introduction and Motivation

For real-world applications, robots need to adapt to unseen and unstructured environments. Thus, developing robust control systems that adapt quickly to new or challenging conditions becomes essential. The surge of data-driven methods and the availability of a higher computation power offers a unique solution. Most of our efforts towards improving robotic systems' adaptation to new environments have been towards developing better controllers. A typical controller-system loop diagram looks like as follows:

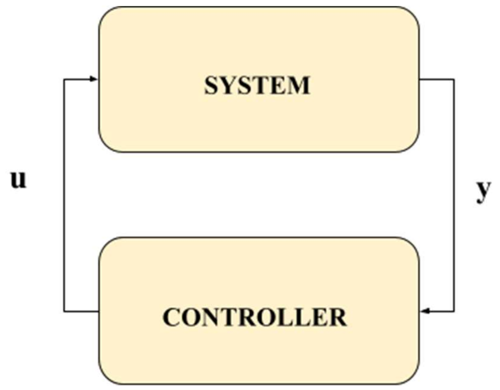


Figure 1 y is the system state estimated by sensor measurement, and u is the controller response to the system state.

However, as we can see in the loop above, there are several areas of research and development apart from the controller design where data-driven methods can prove useful in developing robust and adaptable robotic systems:

a. precise state estimation.
b. optimisation of sensor placement on the system body.
c. system representation through the appropriate dynamical model.

In our work, we have focused on leveraging data-driven techniques for constructing dynamical models which are representative of the robotic system. Conventionally, physics-based equations are employed to model the system dynamics. However, physics-based models present us with a few challenges:

a. modelling of complex nonlinear dynamics
b. precisely accounting for unknown dynamics as in environmental disturbances
c. modelling high-dimensional dynamics with many state inputs

Formulating a precise physics-based model for the above problems becomes a difficult task. However, with the availability of large amounts of data, it becomes imperative to formulate data-driven models of system dynamics. Deep Neural Networks (DNN) have proven helpful in constructing data-driven models. Theoretically, a sufficiently deep neural network can learn even a highly complex functional mapping between the system input and output. DNNs also provide us with the potential to construct generalisable models. Setting this as a precedent, we will build towards developing system identification algorithms utilising deep learning approaches.

Chapter – 2
Background

Sparse Identification of Non-linear Dynamics (SINDy)

SINDy is a data-driven algorithm that employs sparse regression to model system dynamics. There are two features of the SINDy algorithm that are essential to our work –

1. Basis functions:
   Let $x_1, x_2, \ldots\ldots, x_n$ be n state variables of a system. Then, at any time instant t_k, x represents the state of the system where,

$$x^T(t_k) = [x_1(t_k), x_2(t_k), \ldots\ldots, x_n(t_k)]$$

   Suppose, we have data sampled at m such time instants. Then, the information about the state variables can be stored in X such that,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix}$$

   Basis functions are essential features of the SINDy algorithm as they allow interpretability of the learned dynamical equation. The core idea of the basis function is that, the user creates an augmented library of functions, which are used to mature the state variables. Then, these matured features are used to construct dynamical equations. Let us call this augmented library of functions Θ, then SINDy will use the functions in Θ to create a matured set of state variables represented by $\Theta(X)$. This can be seen below:

$$\boldsymbol{\Theta}(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | & | & | & \\ \mathbf{1} & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \sin(2\mathbf{X}) & \cos(2\mathbf{X}) & \cdots \\ | & | & | & | & & | & | & | & | & \end{bmatrix}$$

Figure 2 augmented library consisting of candidate nonlinear functions

   To highlight an instance of how a library function would mature the state features, P_2 represents a polynomial function of degree 2, and its effect on the state feature library X can be seen as follows:

$$\mathbf{X}^{P_2} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_2^2(t_1) & x_2(t_1)x_3(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_2^2(t_2) & x_2(t_2)x_3(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \cdots & x_2^2(t_m) & x_2(t_m)x_3(t_m) & \cdots & x_n^2(t_m) \end{bmatrix}$$

The choice of constructing this set of nonlinear candidate functions is with the user, and it may be unknown to the user what the right choice of candidate functions for a particular problem should be. It highlights the first issue that we will look to address.

2. Sparse Regression:
All the state features may not contribute prominently towards the dynamical model. It is thus imperative that the dynamical equations may not have all the state or control terms present in them. This is highly useful as it would allow us to identify the essential terms contributing to the system dynamics. In turn, this becomes important as it will enable us to assess the effects of changes in particular state features or control inputs on the overall dynamics of the system, allowing greater control over the system.

SINDy models the system dynamics as follows:

$$\dot{X} = \theta(X)\mathcal{E} \text{ (5)}$$

, where $\mathcal{E}$ is the coefficient matrix that helps us analyse the features in $\Theta(X)$ essential to modelling the dynamics. To actively zero out the terms that do not contribute predominantly to the dynamics, we may solve for $\mathcal{E}$ using sparse regression.

Since $\Theta(X)$, in most cases, will not be a square matrix, it makes sense to formulate the solution for $\mathcal{E}$ so as to estimate an $\tilde{\mathcal{E}}$, such that $\tilde{\mathcal{E}}$ minimises the following objective,

$$||\theta(X)\tilde{\mathcal{E}} - \dot{X}||_2 \text{ (6)}$$

To enforce sparsity of feature selection by $\mathcal{E}$ one can either utilize L_1 norm while solving the objective function in (6) or can sequentially threshold the terms of $\mathcal{E}$ to 0. It has been shown that sequential thresholding has similar effects to computing L_1 norm. This algorithm can be modelled as follows:

```
1: while $\mathcal{E}$ does not converge do
2:     Evaluate the feature selection matrix $\mathcal{E}$ as solution to the least-squares
       estimation problem $\|\Theta(X)\mathcal{E} - \dot{X}\|_2$.
3:     To induce sparsity, set $\mathcal{E}$ matrix feature values less than a threshold $\lambda$
       to 0.
4:     Drop the features corresponding to the thresholded values of $\mathcal{E}$ in
       $\Theta(X)$.
5: end while
```

Algorithm 1 Least-squares Sparse Regression

The above algorithm is helpful to estimate $\mathcal{E}$ in the cases where the basis functions are fully-formed. However, in the instances of an evolving basis function, finding definite solutions to the above mentioned least-squares estimation problem can guide the evolution of the basis function in the wrong direction. Then the basis function may cease to be a misguided representation of data and may not evolve fully to understand the inherent dynamics of the system.

Model-Agnostic Meta Learning (MAML)

Meta Learning is a mechanism for learning to learn. An example that could be used to highlight what we mean by the above statement is that we can train a quadrotor controller model to fly under specific wind disturbances, but when the quadrotor learns to compensate for any such wind disturbance by learning not the techniques to compensate for a specific wind but by learning "how to compensate", the model is said to meta learn.

Model-Agnostic meta learning framework looks to leverage meta-learning approaches to all tasks despite the nature of learning framework and the architecture of the neural network model.

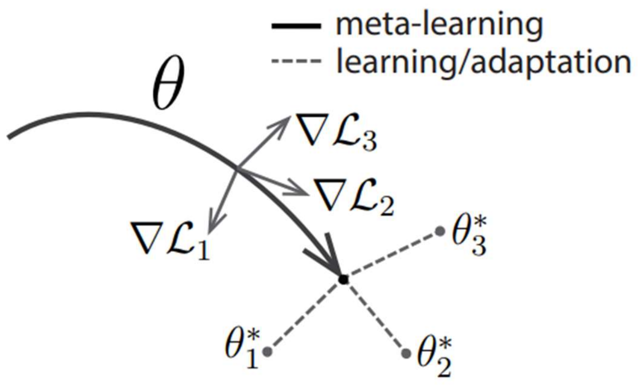


Figure 3 MAML algorithm representation for optimizing Θ to enable quick adaptation to new tasks

The goal of the MAML algorithm is to make the model parameters susceptible to fast adaptation when it encounters a new task from a similar distribution. As can be seen in Figure 3, the algorithm drives the model parameter Θ towards an optimum point from where it can be quickly optimized to $\Theta_k^\star$ to adapt to a new unseen task k.

The algorithm works by first sampling tasks from a given distribution, and meta-training the model over those tasks. The algorithm relies on gradient descent for optimizing its parameters. The MAML algorithm can be written as follows:

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha, \beta$: step size hyperparameters
 1: randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_i$ do
 5:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
 6:         Compute adapted parameters with gradient de-
            scent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
 7:     end for
 8:     Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
 9: end while
```

Algorithm 2: The MAML Algorithm

In the algorithm, the loss function $\mathcal{L}$ is dependent on the learning framework used. For regression tasks, as we will use in our case, we will formulate $\mathcal{L}$ as a mean squared error loss function. Also, while the hyperparameter α is used to update the model parameter after calculating the loss for each task, the hyperparameter β updates the model parameter after evaluating the loss term over all the tasks used for training.

Baseline Work

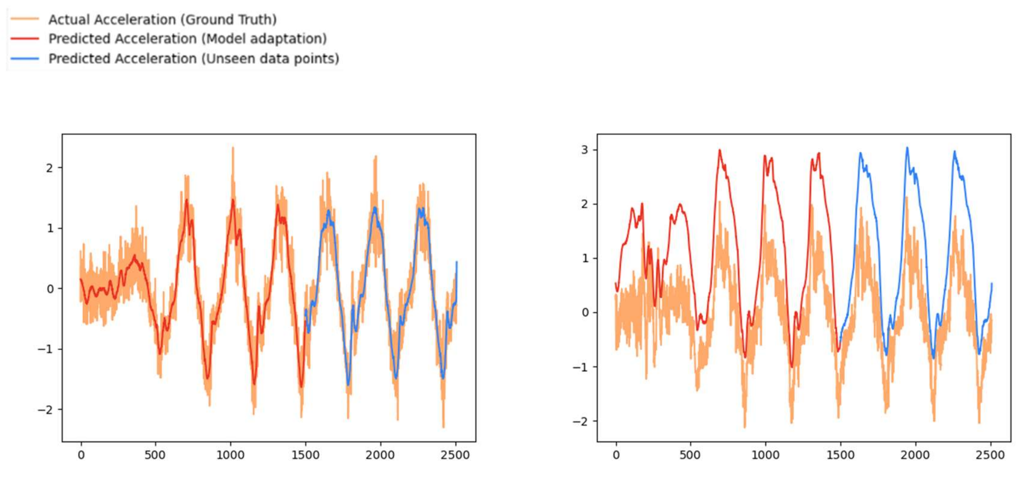
The challenge posed by not knowing the right set of library basis functions motivates us to utilize a DNN as a basis function. DNNs have been useful in modelling the dynamical equations, and can theoretically learn even a sufficiently complex mapping.

In the work going forward, we will try to learn the translational partial dynamics of the quadrotor, where the acceleration will be modelled as follows:

$$\dot{v} = f(v, Rf_u)$$

The equation has been discussed in detail in Chapter 3. Here, we will show the performance of a simple fully connected DNN with ReLU activations in modelling the dynamics. While a DNN may learn to predict the dynamical behaviour in a certain condition, it may not necessarily generalize well under different environmental conditions, which pose a domain shift.

It can be understood from the comparative analysis below. We have presented the performance of a DNN which was trained to predict the x-component of acceleration of a quadrotor under the wind influence of 4.2 m/s and was tested under the wind speed of 8.5 + 2.4 sin(t) m/s.



Prediction under 4.2 m/s wind

Prediction under 8.5 + 2.4 sin(t) m/s wind

It is clear from the above results that a naïve DNN is not able to generalize well in case of a domain shift in the distribution of data. This necessitates our attempt at meta-learning the basis library functions.

Also, from rigorous experimentation, we have seen that for our dataset, sin(x) and cos(x) suffice as a set of basis functions to model the dynamics using the SINDy algorithm. Therefore, any comparative results presented ahead will use these as library functions while training the SINDy model.

Chapter - 3
Problem Formulation and Methodology

The following equations define the kinematics and dynamics of a quadrotor:

$$\dot{p} = v \text{ (1)}$$

$$m\dot{v} = mg + Rf_u + f_a \text{ (2)}$$

$$\dot{R} = RS(\omega) \text{ (3)}$$

$$J\dot{\omega} = J\omega \times \omega + \tau_u + \tau_a \text{ (4)}$$

, where p is the global position vector, v is the velocity vector, m is the mass, J is the inertia matrix, g is gravitational acceleration, R is the rotation matrix such that $R \in SO(3)$, f_a is external disturbance force, $S(\omega)$is the skew-symmetric matrix of angular velocity ω, τ_u and τ_a are control and external disturbance torques, respectively.

We aim to learn the partial dynamics equations where we will use the following formulation of the translational dynamics:

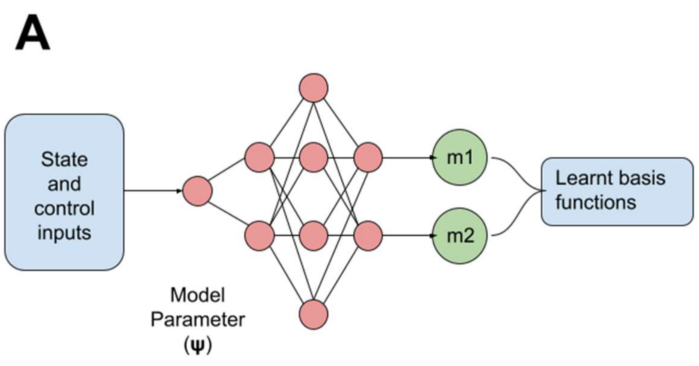
$$\dot{v} = f(v, Rf_u)$$

where v is the translational velocity of the quadrotor and Rf_u is the control input, where $f_u^T = [0,0,\mathrm{T}]$, where T is the magnitude of thrust generated at any instant. We use the above mapping to learn the partial dynamics.

As motivated by the baseline work, we will aim to meta-learn a pair of basis functions to model the dynamics. The motivation to limit ourselves to a pair of candidate library functions comes from the fact that only two basis functions: sin(x) and cos(x) were sufficient while training the SINDy model.

In order to achieve these set of basis functions, we will use a single fully-connected DNN with two output features. As labelled in Figure 5A these two output features m1 and m2 will serve as our basis functions. We employ a single DNN while modelling the basis functions to encourage the learning of cross-interaction terms between the features and the basis functions if any. Cross-interaction terms are commonly seen while modelling complex nonlinear dynamics, and this further justifies our choice of modelling the basis functions utilising a single DNN. We will represent the model parameters of this DNN by **ψ.**

For training our proposed algorithm, we will use the Neural Fly dataset. For meta-training, we will use the data gathered by a baseline nonlinear controller for a quadrotor



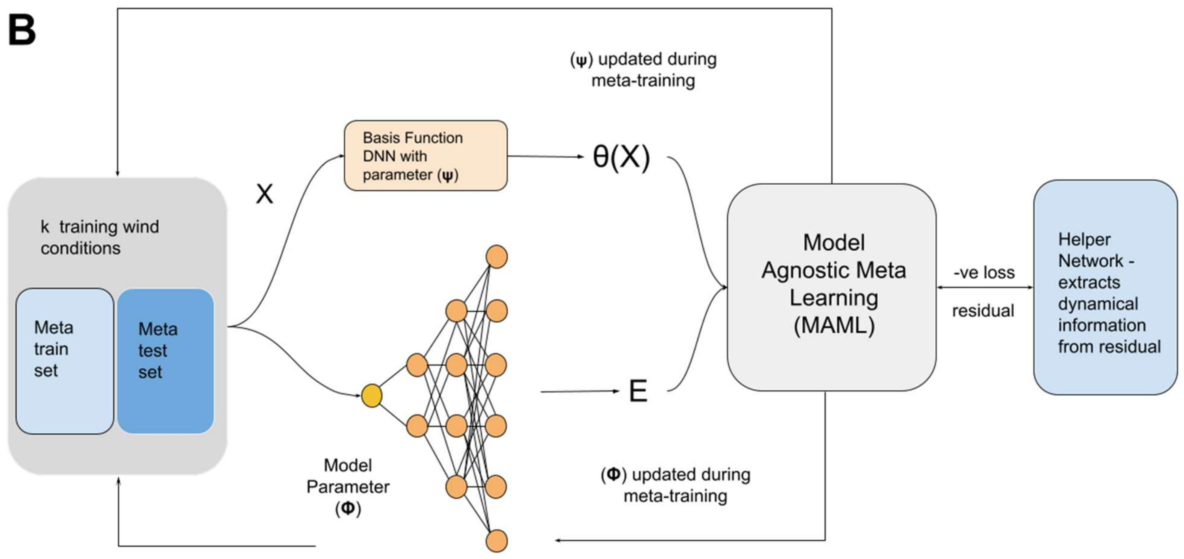

B
(ψ) updated during meta-training
k training wind conditions
Meta train set
Meta test set
X
Basis Function DNN with parameter (ψ)
θ(X)
E
Model Agnostic Meta Learning (MAML)
-ve loss
residual
Helper Network - extracts dynamical information from residual
Model Parameter (Φ)
(Φ) updated during meta-training

Figure 4 A. Modelling the basis functions as two output features of a DNN, m1 and m2 being the basis features. B. The model architecture and the training flow chart, model parameters ***Φ*** *and* ***ψ*** *are updated cyclically, the negative loss of the helper network is backpropagated to make the residual devoid of any dynamical information and help guide the basis functions towards understanding the dynamical features.*

following a randomised trajectory under the following wind conditions:

a. 0 m/s wind
b. 1.3 m/s wind
c. 2.5 m/s wind
d. 3.7 m/s wind
e. 4.9 m/s wind
f. 6.1 m/s wind

The algorithm proposed is as follows:

```
Require: $v(W), Rfu(W)$: distribution over training wind data
Require: $a, B$: step size hyperparameters
 1: randomly initialize $\phi$ and $\psi$, model parameter of the DNN giving us $\mathcal{E}$
    and the basis library functions in $\Theta$ respectively
 2: while not done do
 3:   Sample batch of tasks $v_i, Rfu_i \sim v(W), Rfu(W)$
 4:   for all $v_i, Rfu_i$ do
 5:     Evaluate the feature matrix $\Theta_\psi(v_i, Rfu_i)$
 6:     Evaluate the feature selection matrix $\mathcal{E} = \phi(v_i, Rfu_i)$
 7:     To induce sparsity, set $\mathcal{E}$ values less than a threshold $\lambda$ to 0
 8:     Evaluate the loss $L_i = [\dot{v}_i - \Theta(X) * \mathcal{E}]^2$ which takes the form:
            $$L_i = [\dot{v}_i - \Theta_\psi(v_i, Rfu_i) * \phi(v_i, Rfu_i)]^2$$
 9:     for even epochs do
10:       Evaluate $\nabla_\phi L_i(f_\phi)$ using $v_i, Rfu_i$
11:       Compute updated $\phi$ using gradient descent:
            $$\phi' \leftarrow \phi - \alpha * \nabla_\phi L_i(f_\phi)$$
12:     end for
13:     for odd epochs do
14:       Evaluate $\nabla_\psi L_i(f_\psi)$ using $v_i, Rfu_i$
15:       Compute updated $\psi$ using gradient descent:
            $$\psi' \leftarrow \psi - \alpha * \nabla_\psi L_i(f_\psi)$$
16:     end for
17:     Sample datapoints $D = \{v_i, Rfu_i\}$ from $v(W), Rfu(W)$ for the
        meta-update
18:   end for
19:   Update using $D$:
            $$\phi \leftarrow \phi - \beta * \nabla_\phi \Sigma_{v_i, Rfu_i \sim v(W), Rfu(W)} L_i(f'_\phi)$$
            $$\psi \leftarrow \psi - \beta * \nabla_\psi \Sigma_{v_i, Rfu_i \sim v(W), Rfu(W)} L_i(f'_\psi)$$
20: end while
```

Algorithm 3: The proposed meta-learning algorithm for system identification.

The model architecture and the training loop presented in Figure 4B will take the input from the data sampled from the above-mentioned experiments. Since we are modelling the dynamics as a function of the quadrotor state v and the control input Rf_u, we will pass these two features as an augmented input to both our DNNs: the basis function DNN with model parameter $\boldsymbol{\psi}$ and the feature selection DNN with model parameter $\boldsymbol{\Phi}$**.** Note that to model the dynamics we are using the dynamical formulation as in the SINDy algorithm:

$$\dot{X} = \theta(X)\mathcal{E}$$

Our goal would be to learn the acceleration pattern which will be equivalent to $\dot{X}$ in the above equation. Our state input feature X will be the velocity of the quadrotor and Rf_u will serve as the control input, which will also be passed augmented to the state parameter. Our basis function DNN ($\boldsymbol{\psi}$) will model an equivalent of $\theta(X)$ and the feature selection DNN ($\boldsymbol{\Phi}$) will model an equivalent of $\mathcal{E}$. Both the model parameters $\boldsymbol{\psi}$ and $\boldsymbol{\Phi}$ would be meta-trained utilising the Algorithm 3.

For an improved training with a better understanding of the inherent system dynamics and later adaptation to unseen tasks, the helper network becomes an essential complement to Algorithm 3. The helper network taken the residual dynamics left after going through the training loop in Algorithm 3 as input. In our case, the residual dynamics is simply the difference between the predicted acceleration by our algorithm and the ground truth.

The job of the helper network is two-fold:

a. To predict the ground truth acceleration from the residual inputs. Over time, the helper network gets better at predicting the ground truth acceleration.

b. To provide feedback to the rest of the architecture regarding their performance.

This goal of the feedback is to help the rest of the architecture calibrate its performance. Since, the final aim is to accurately predict the acceleration, the residual left after the prediction should ideally be devoid of any information about the dynamics of the system. In that case, the performance of the helper network will be poor. And that formulates our goal: to make the algorithm performance better while making the helper performance worse over time. And since, the helper network is optimized to get pretty well at its job over the training process this leads to a tug-of-war which eventually guides the training of the model architecture in the right direction towards developing an understanding of the dynamical features. To achieve this the negative of the performance measure (loss) is added to the overall loss in Algorithm 3, which gets modified as follows:

$$L_i(f) = L_i(f) - \mathcal{L}$$

, where $\mathcal{L}$ is the mean squared loss of the helper network prediction. It can be seen from the equation above that larger the value of $\mathcal{L}$ lower will be the overall loss term, which will guide the model to understand dynamics better.

Once, we have the meta-trained parameters, these parameters can be quickly adapted to do well on any new task by optimizing the parameters using gradient descent. Moving ahead we will show a comparative performance analysis of our algorithm with SINDy, both quantitative and qualitative, in the following wind conditions:

1. 4.2 m/s wind
2. 8.5 m/s wind
3. 12.1 m/s wind
4. 8.5 + 2.4 sin(t) m/s wind

Chapter – 4

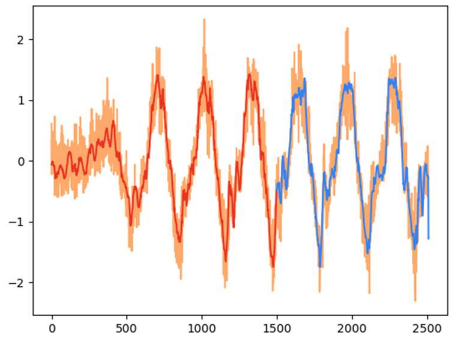
Results and Discussions

For adaptation to specific wind conditions equal number of training epochs were used.

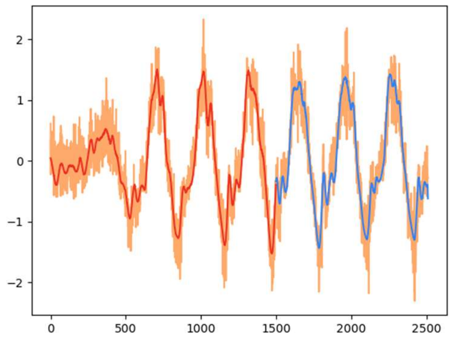
We will use the following colour scheme in presenting the results:

— Actual Acceleration (Ground Truth)
— Predicted Acceleration (Model adaptation)
— Predicted Acceleration (Unseen data points)

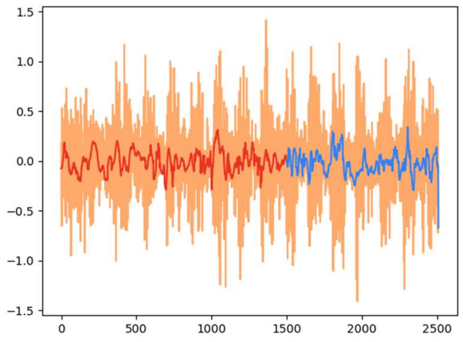
4.1 Model vs SINDy performance under 4.2 m/s wind:



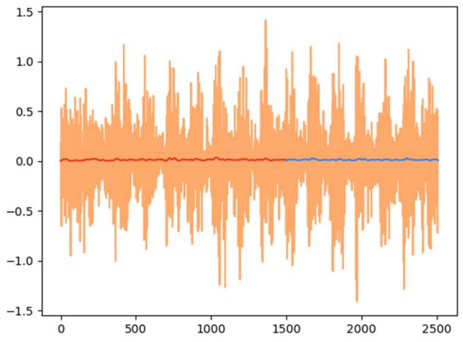
Model acceleration prediction in x



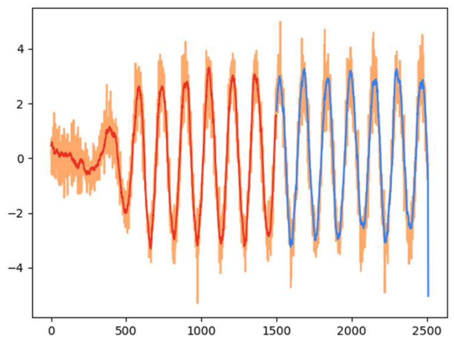
SINDy acceleration prediction in x



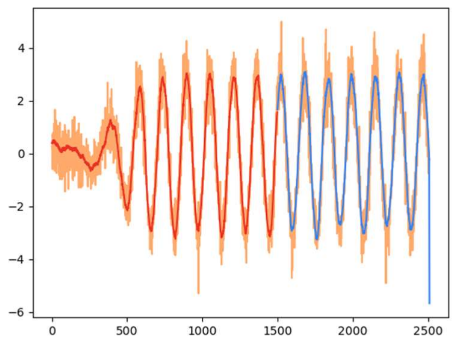
Model acceleration prediction in y



SINDy acceleration prediction in y

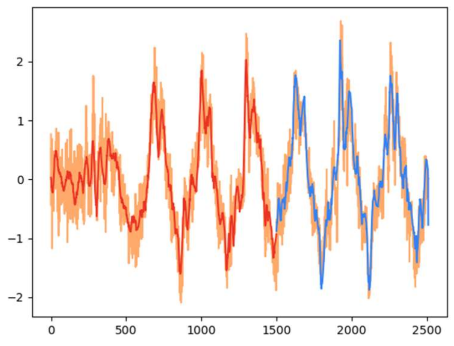


Model acceleration prediction in z

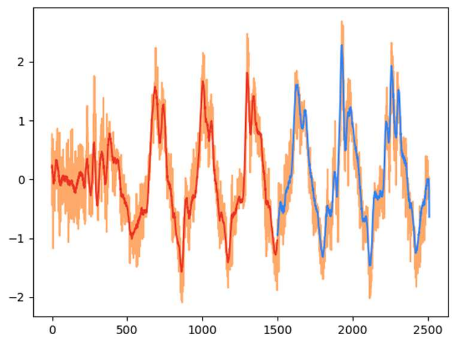


SINDy acceleration prediction in z

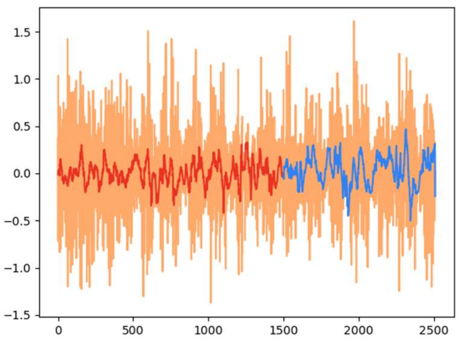
4.2 Model vs SINDy performance under 8.5 m/s wind:



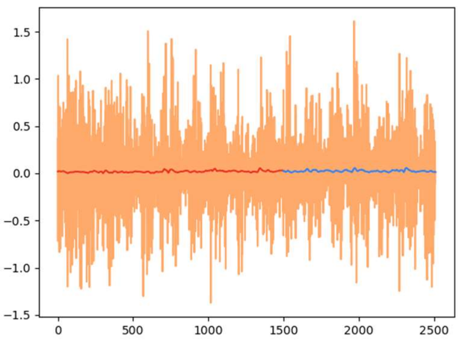
Model acceleration prediction in x



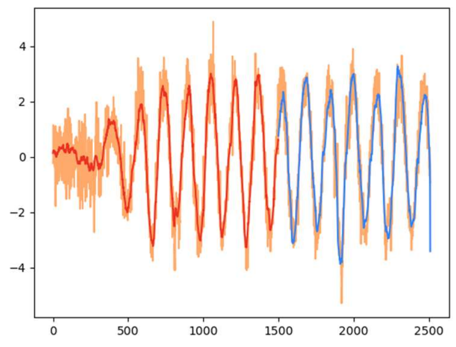
SINDy acceleration prediction in x



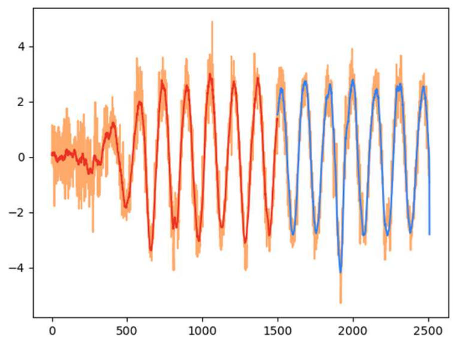
Model acceleration prediction in y



SINDy acceleration prediction in y

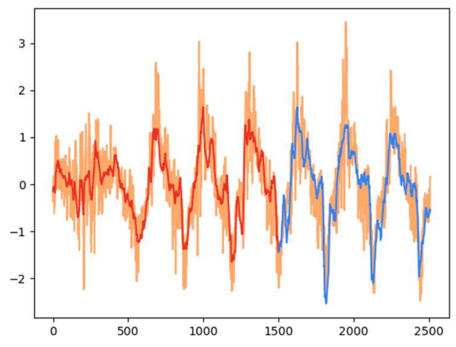


Model acceleration prediction in z

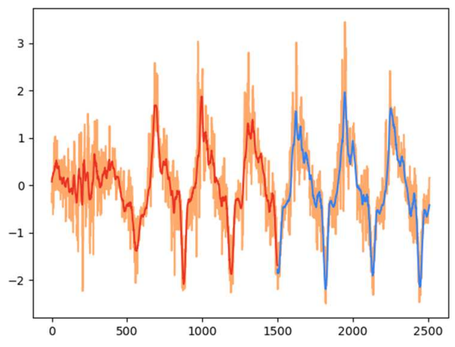


SINDy acceleration prediction in z

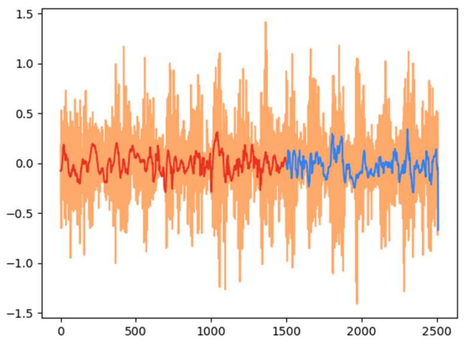
4.3 Model vs SINDy performance under 12.1 m/s wind:



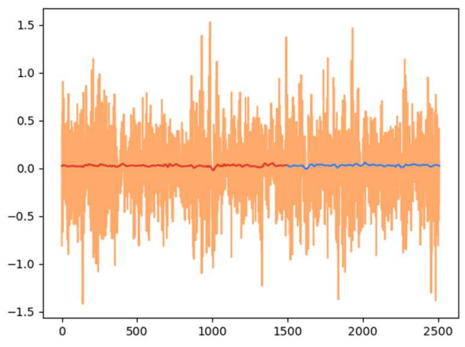
Model acceleration prediction in x



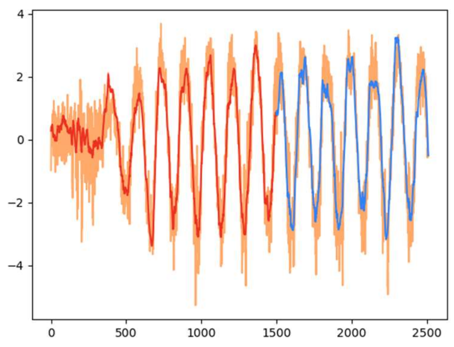
SINDy acceleration prediction in x



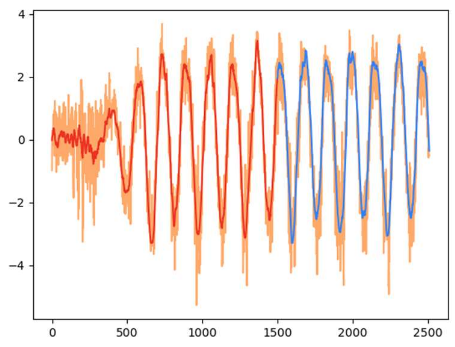
Model acceleration prediction in y



SINDy acceleration prediction in y

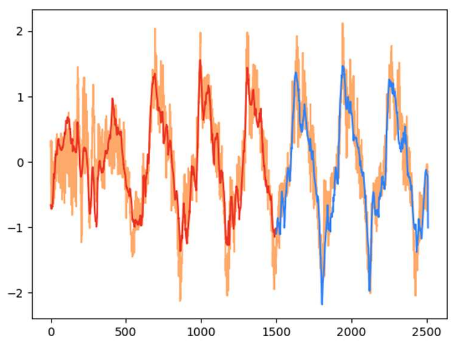


Model acceleration prediction in z

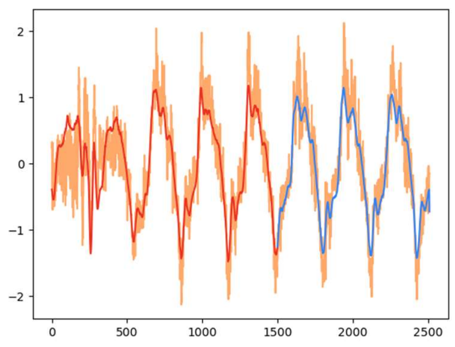


SINDy acceleration prediction in z

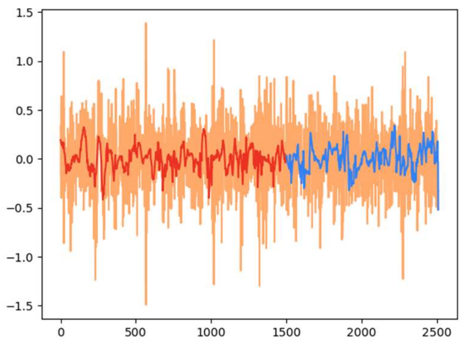
4.4 Model vs SINDy performance under 8.5 + 2.4 sin(t) m/s wind:



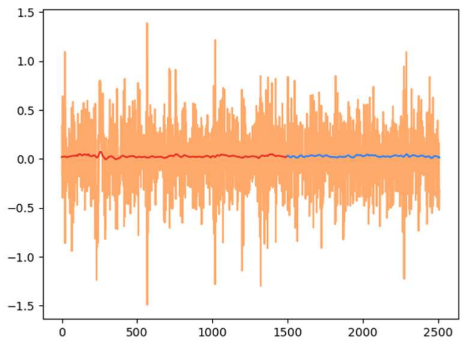
Model acceleration prediction in x



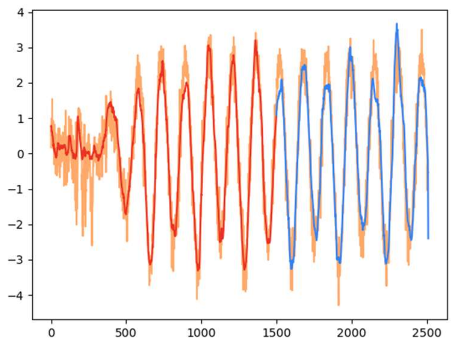
SINDy acceleration prediction in x



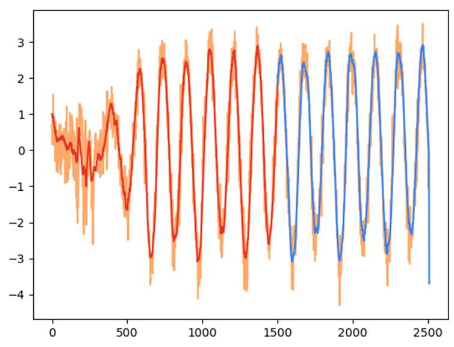
Model acceleration prediction in y



SINDy acceleration prediction in y



Model acceleration prediction in z



SINDy acceleration prediction in z

For SINDy model, sin(x) and cos(x) were used as library functions.

4.5 Quantitative analysis of model performance against SINDy

SINDy (MSE)	Training Error (online adaptation)	Testing Error (generalization error)	Averaged Error
4.2 m/s wind	0.155732	0.194475	0.171331
8.5 + 2.4 sin(t) m/s wind	0.138483	0.104922	0.124970
8.5 m/s wind	0.193284	0.171666	0.184580
12.1 m/s wind	0.226771	0.232093	0.228914

Model (MSE)	Training Error (online adaptation)	Testing Error (generalization error)	Averaged Error
4.2 m/s wind	0.156434	0.208611	0.177442
8.5 + 2.4 sin(t) m/s wind	0.156630	0.149163	0.153624
8.5 m/s wind	0.209953	0.218615	0.213440
12.1 m/s wind	0.268037	0.302406	0.281875

CHAPTER - 5

Conclusion and Future Work

In the presented work, we have successfully shown that our proposed algorithm to meta-learn the basis functions and the feature selection matrix for nonlinear system identification adapts and generalises well to model dynamical equations under unseen wind conditions prior to its testing phase. It can also be seen that the mean squared error of the predictions of our algorithms lies in the range of 5-30% of the state-of-the-art SINDy algorithm, depending upon the complexity of the wind pattern. However, our model provides two benefits over the SINDy algorithm:

1) The user need not have domain specific knowledge required to choose the right set of basis library functions as in the SINDy algorithm.

2) It gives us a generalizable set of parameters for our basis functions, which can be optimized to quickly adapt to unseen conditions.

3) Also, even if we encounter conditions which are totally out of distribution on which the model was trained, since model agnostic meta-learning is based on gradient descent, it provides us with a theoretical guarantee that with sufficient number of gradient steps, our model will be able to adapt to even the most challenging out-of-distribution conditions.

Our present work sets up the groundwork to address the following challenging in robotic system identification in future:

1) Modelling of the entire dynamical structure of the quadrotor, incorporating the rotational dynamics terms and their interactions with translational dynamics.

2) Learning the residual dynamics representation, which is an effect of the system response to the dynamics. Thus, laying the foundation for system-oriented dynamical models with an in-built understanding of the robotic system.

REFERENCES

1. Data-driven discovery of coordinates and governing equations
2. Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds
3. Discovering governing equations from data: Sparse identification of nonlinear dynamical systems
4. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks
5. Neural Lander: Stable Drone Landing Control using Learned Dynamics
6. Neural Networks in Optimization
7. Neural-Swarm: Decentralized Close-Proximity Multirotor Control Using Learned Interactions
8. Domain-Adversarial Training of Neural Networks
9. A Brief Review of Domain Adaptation
10. Deep Unsupervised Domain Adaptation: A Review of Recent Advances and Perspectives